What is Brain Stroke ??

1. Brain Stroke happens when there is a blockage in the blood circulation in the brain or when a blood vessel in the brain breaks and leaks. The burst or blockage prevents blood and oxygen reaching the brain tissue. Without oxygen the tissues and cells in the brain are damaged and die in no time leading to many symptoms.
2. Once brain cells die, they generally do not regenerate and devastating damage may occur, sometimes resulting in physical, cognitive and mental disabilities. It is crucial that proper blood flow and oxygen be restored to the brain as soon as possible.

Context

1. According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.
2. This dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status.
3. Each row in the data provides relavant information about the patient.

Attribute Information

1. Gender: "Male" or "Female"
2. Age: Age of the patient
3. Hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
4. Heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
5. Ever_married: "No" or "Yes"
6. Work_type: "Children", "Govt_job", "Never_worked", "Private" or "Self-employed"
7. Residence_type: "Rural" or "Urban"
8. Avg_glucose_level: Average glucose level in blood
9. Bmi: Body Mass Index
10. Smoking_status: "Formerly smoked", "Never smoked", "Smokes" or "Unknown"*
11. Stroke: "Yes" if the patient had a stroke or "No" if the patient not had a stroke

*Note: "Unknown" in Smoking_status means that the information is unavailable for this patient

Data Preparation/Cleaning

1. As the dataset from kaggle was not very suitable for data analysis, we had to change the format of some data in the dataset. We also had to do separate data preparation for exploratory analysis and machine learning.

2. Some of our data preparation were:
    - Treating NaN values to the integer.
    - Performing visaualization techniques.
    - Scaling the numerical values for better accuracy.
    - Encoding the categorical variables so that it can be fit into the classifier model.
    - Separating the data into Predicted and Target variables.
    - Separating the data into training and testing sets(Training Sets: 75% ,Testing Sets: 25%).

Python Libraries Use For Analysis

1. Numpy = Use for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions.
2. Pandas = It is mainly used for data analysis and associated manipulation of tabular data in DataFrames. IT allows importing data from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel.
3. Matplotlib = Visualizations are the easiest way to analyze and intake information.pyplot is used to create figures and change their characteristics.
4. Seaborn = Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.
5. Plotly = It is great for building beautiful and interactive visualizations. It is an awesome tool for discovering patterns in a dataset before delving into machine learning modeling.
6. Scikit Learn = It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Selection of classification model

- Classification models are a subset of supervised machine learning.
- It reads some input and generates an output that classifies the input into some category.
- Used to forecast or classify the distinct values such as Real or False, Male or Female, Spam or Not Spam, etc.
- The classification models are built with unordered values with dependent variables.
- The task of the classification algorithm is to map the input value(x) with the discrete output variable(y).

Classifier Models

- DecisionTree Classifier
- RandomForest Classifier
- KNeighbors Classifier
- GaussianNB Classifier
- Support Vector Classification

- AdaBoost Classifier

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

Load the Dataset

In [2]:
```
df=pd.read_csv('Brain-Stroke-Prediction.csv')
df
```

Out[2]:

	Gender	Age	Hypertension	Heart_disease	Ever_married	Work_type	Residence_type	Avg_g
0	Male	67.0	0	1	Yes	Private	Urban	
1	Female	61.0	0	0	Yes	Self-employed	Rural	
2	Male	80.0	0	1	Yes	Private	Rural	
3	Female	49.0	0	0	Yes	Private	Urban	
4	Female	79.0	1	0	Yes	Self-employed	Rural	
...	...	...	...	...	...	...	...	...
5104	Female	80.0	1	0	Yes	Private	Urban	
5105	Female	81.0	0	0	Yes	Self-employed	Urban	
5106	Female	35.0	0	0	Yes	Self-employed	Rural	
5107	Male	51.0	0	0	Yes	Private	Rural	
5108	Female	44.0	0	0	Yes	Govt_job	Urban	

5109 rows × 11 columns

Shape of our dataset

In [3]:
```
print(f'The training set contains {df.shape[0]} rows and {df.shape[1]} columns.')
```

```
The training set contains 5109 rows and 11 columns.
```

- Observation:This dataset consist of 5109 records and 11 features

In [4]:
```
df.head()
```

Out[4]:

	Gender	Age	Hypertension	Heart_disease	Ever_married	Work_type	Residence_type	Avg_gluc
0	Male	67.0	0	1	Yes	Private	Urban	
1	Female	61.0	0	0	Yes	Self-employed	Rural	
2	Male	80.0	0	1	Yes	Private	Rural	
3	Female	49.0	0	0	Yes	Private	Urban	
4	Female	79.0	1	0	Yes	Self-employed	Rural	

- Observation:Printing the top most 5 values from the dataset.

Treating Null Values

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Gender               0
Age                  0
Hypertension         0
Heart_disease        0
Ever_married         0
Work_type            0
Residence_type       0
Avg_glucose_level    0
Bmi                201
Smoking_status       0
Stroke               0
dtype: int64
```

- Observation:
- Total Null Values in the dataset is 201.

Filling Null Values

In [6]:
```
df['Bmi'].fillna(df['Bmi'].mean(),inplace=True)
```

- Observation:The Null Values are filled with Mean values

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Gender               0
Age                  0
Hypertension         0
Heart_disease        0
Ever_married         0
Work_type            0
Residence_type       0
Avg_glucose_level    0
Bmi                  0
Smoking_status       0
Stroke               0
dtype: int64
```

- Observation:There is no null values in this dataset

Information about Features

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5109 entries, 0 to 5108
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             5109 non-null   object 
 1   Age                5109 non-null   float64
 2   Hypertension       5109 non-null   int64  
 3   Heart_disease      5109 non-null   int64  
 4   Ever_married       5109 non-null   object 
 5   Work_type          5109 non-null   object 
 6   Residence_type     5109 non-null   object 
 7   Avg_glucose_level  5109 non-null   float64
 8   Bmi                5109 non-null   float64
 9   Smoking_status     5109 non-null   object 
 10  Stroke             5109 non-null   object 
dtypes: float64(3), int64(2), object(6)
memory usage: 439.2+ KB
```

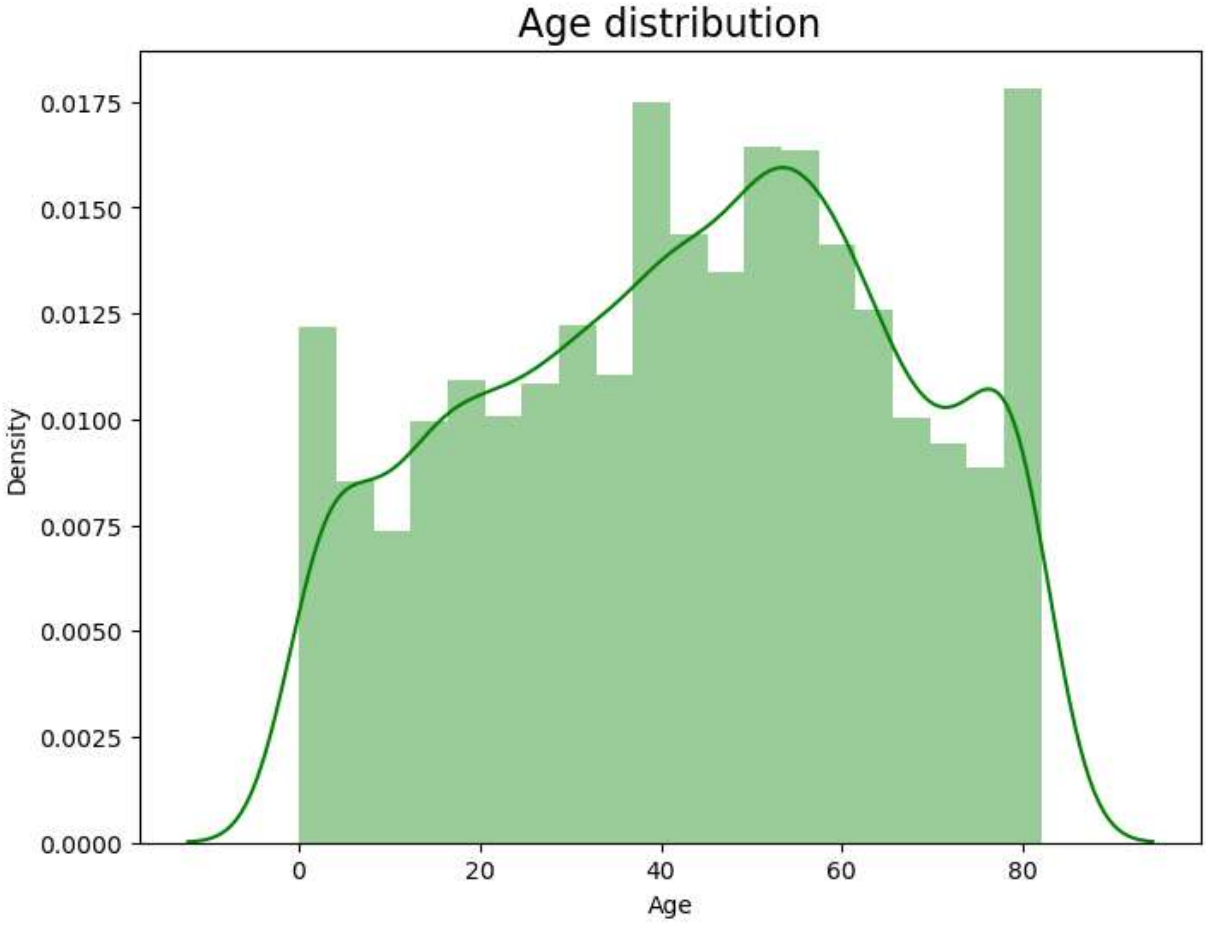
- Observation:It contains the number of columns,column labels,column data types,memory usage,range index and the number of cells in each column(non-null values).

Exploratory Data Analysis

Distplot

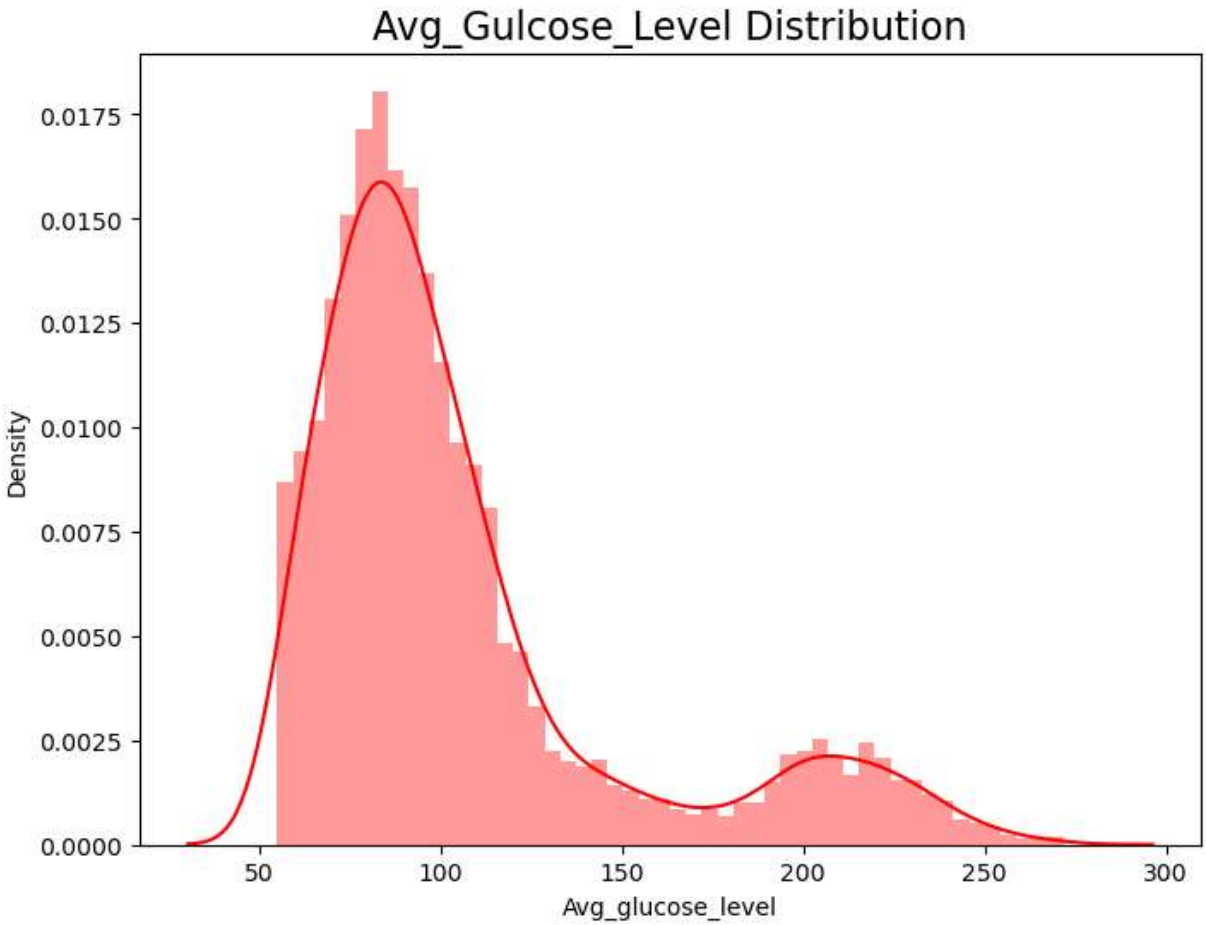
- A Distplot depicts the variation in the data distribution.
- It represents the overall distribution of continuous data variables.
- It accepts the data variable as an argument and returns the plot with the density distribution.

In [9]:
```
plt.figure(figsize = (8, 6))
ax = sns.distplot(df['Age'],color = 'green')
ax.set_title('Age distribution', fontdict = {'fontsize': 16})
plt.show()
```



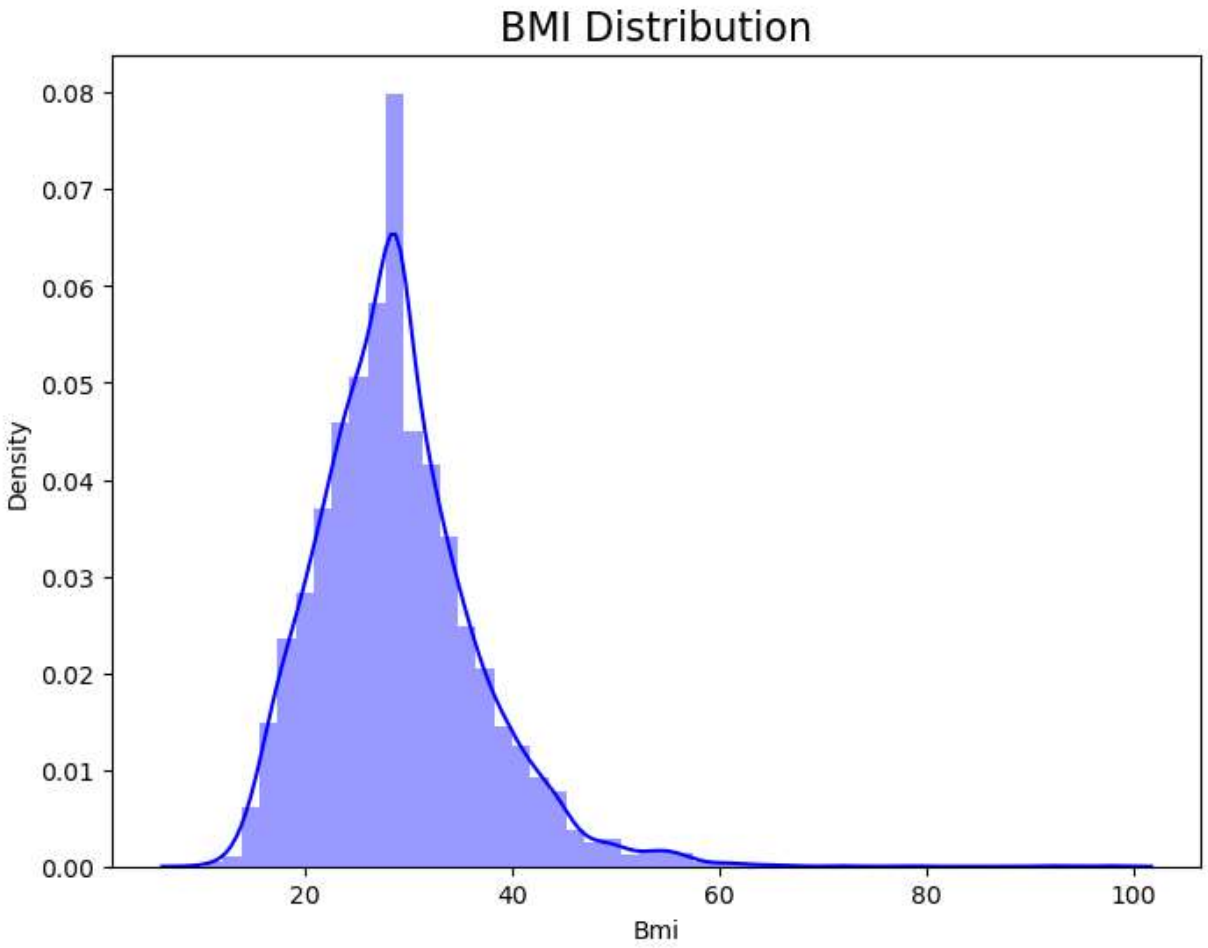
- Observation:A major number of observations are between their 40's and 60's.

In [10]:
```
plt.figure(figsize = (8, 6))
ax = sns.distplot(df['Avg_glucose_level'],color = 'red')
ax.set_title('Avg_Gulcose_Level Distribution', fontdict = {'fontsize': 16})
plt.show()
```



- Observation:The majority of observations have an average Glucose level of 60-120, but very few of them have that of 120-250.

In [11]:
```
plt.figure(figsize = (8, 6))
ax = sns.distplot(df['Bmi'],color = 'blue')
ax.set_title('BMI Distribution', fontdict = {'fontsize': 16})
plt.show()
```



- Observation:The majority of observations have a bmi of 25-35, but very few of them have that of 50-80

Countplot

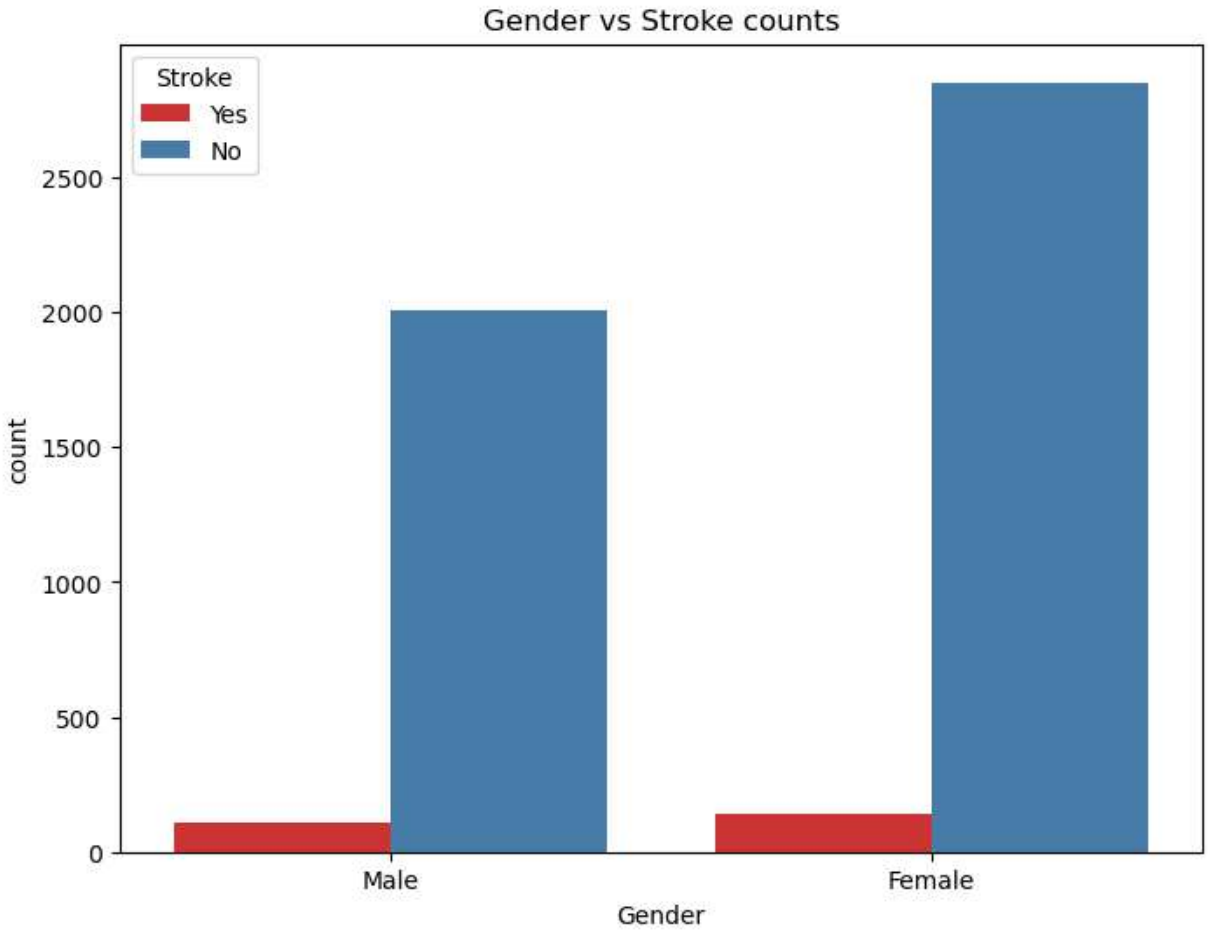
- countplot()Show the counts of observations in each categorical bin using bars.
- It returns the count of total values for each category using bars.

Difference between countplot and barplot

- Bar plots look similar to count plots, but instead of the count of observations in each category, they show the mean of a quantitative variable among observations in each category.

Gender vs Stroke counts

In [12]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Gender'], hue = df['Stroke'])
ax.set_title('Gender vs Stroke counts')
plt.show()
```



In [13]:
```
pd.crosstab(
    index = df['Gender'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

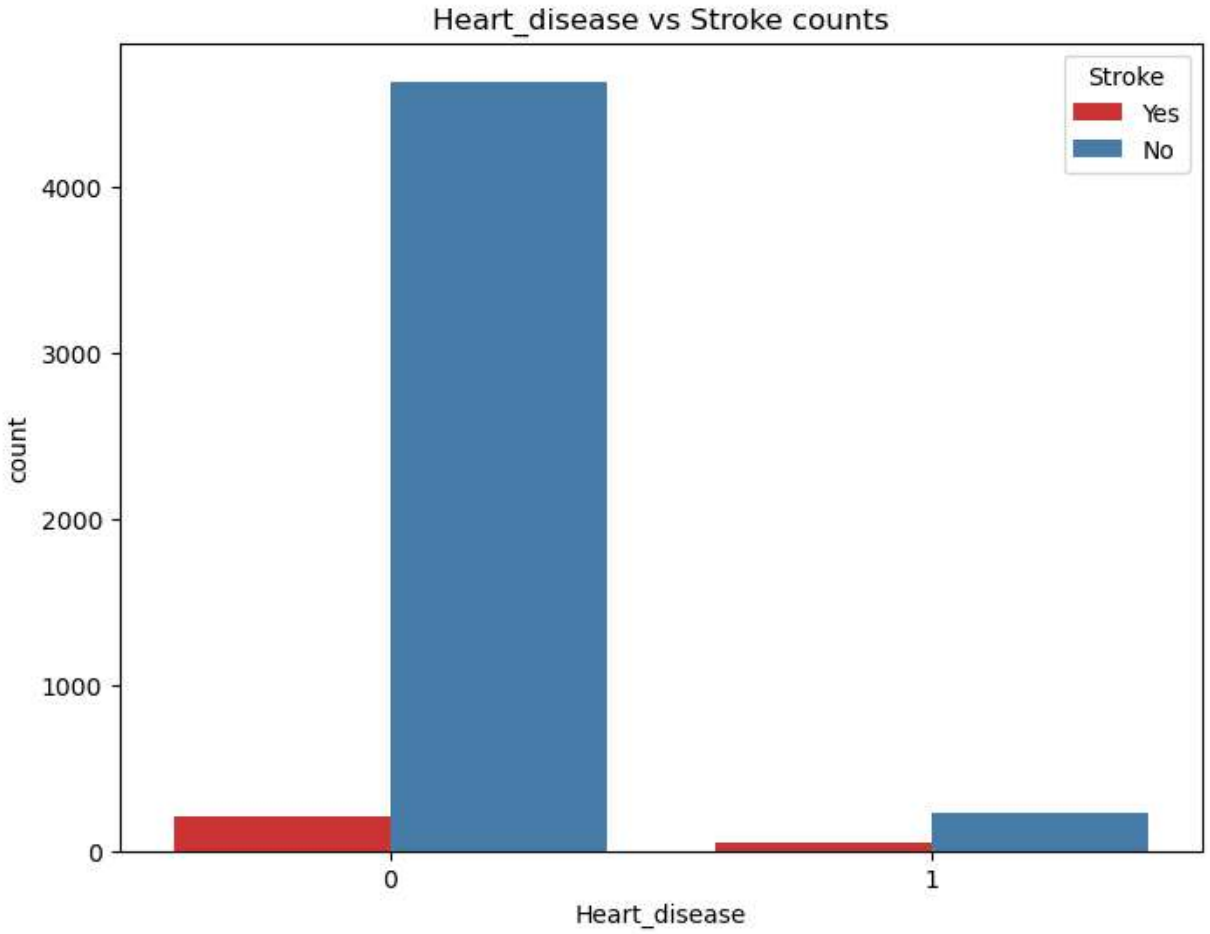
Out[13]:

Stroke	No	Yes
Gender		
Female	0.952906	0.047094
Male	0.948936	0.051064
All	0.951262	0.048738

- Observation:By comparing both male and female,males are mostly afected by stroke.

Heart_disease vs Stroke counts

In [14]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Heart_disease'], hue = df['Stroke'])
ax.set_title('Heart_disease vs Stroke counts')
plt.show()
```



In [15]:
```
pd.crosstab(
    index = df['Heart_disease'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

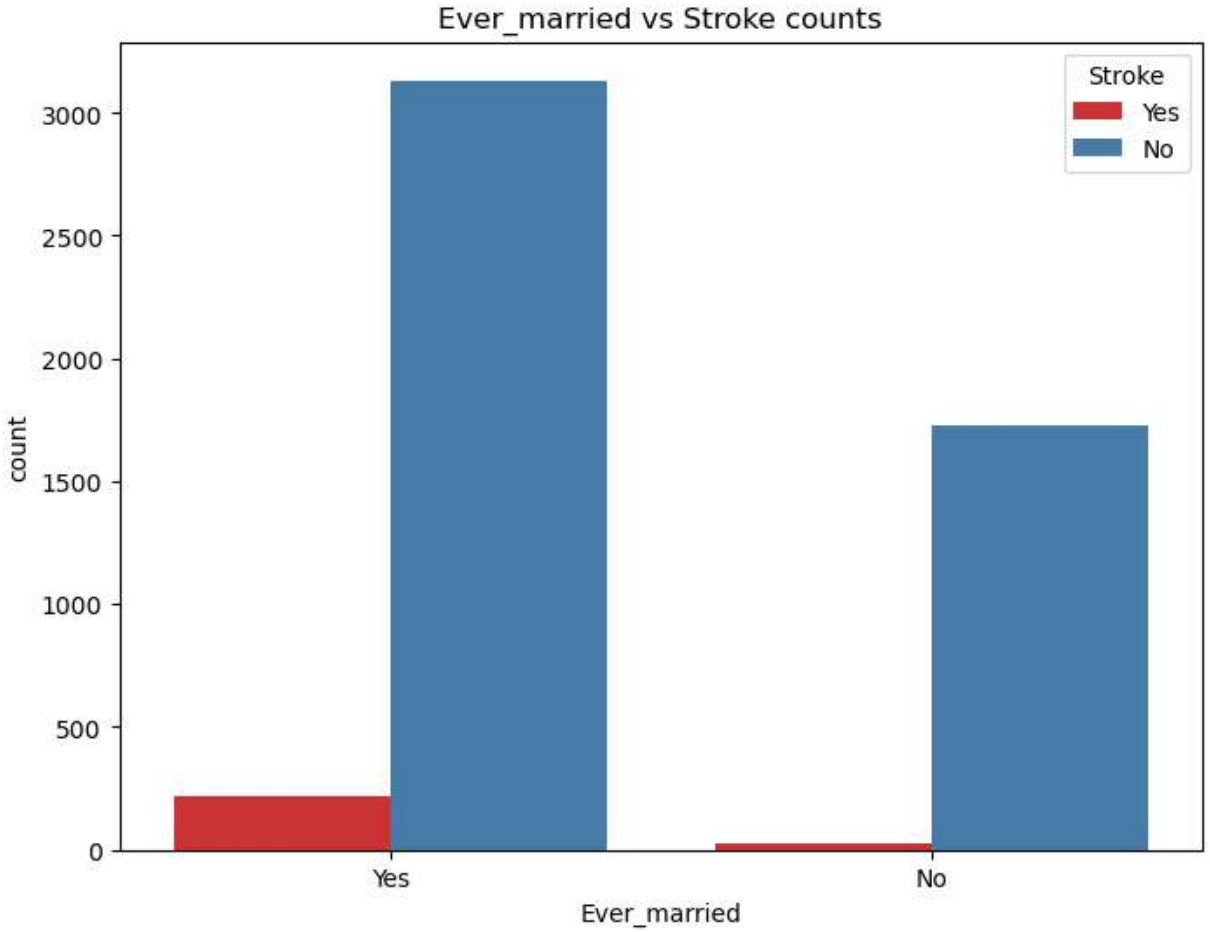
Out[15]:

Stroke	No	Yes
Heart_disease		
0	0.958204	0.041796
1	0.829710	0.170290
All	0.951262	0.048738

- Observation:Majority of heart disease persons are affected by brain stroke.

Ever_married vs Stroke counts

In [16]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Ever_married'], hue = df['Stroke'])
ax.set_title('Ever_married vs Stroke counts')
plt.show()
```



In [17]:
```
pd.crosstab(
    index = df['Ever_married'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

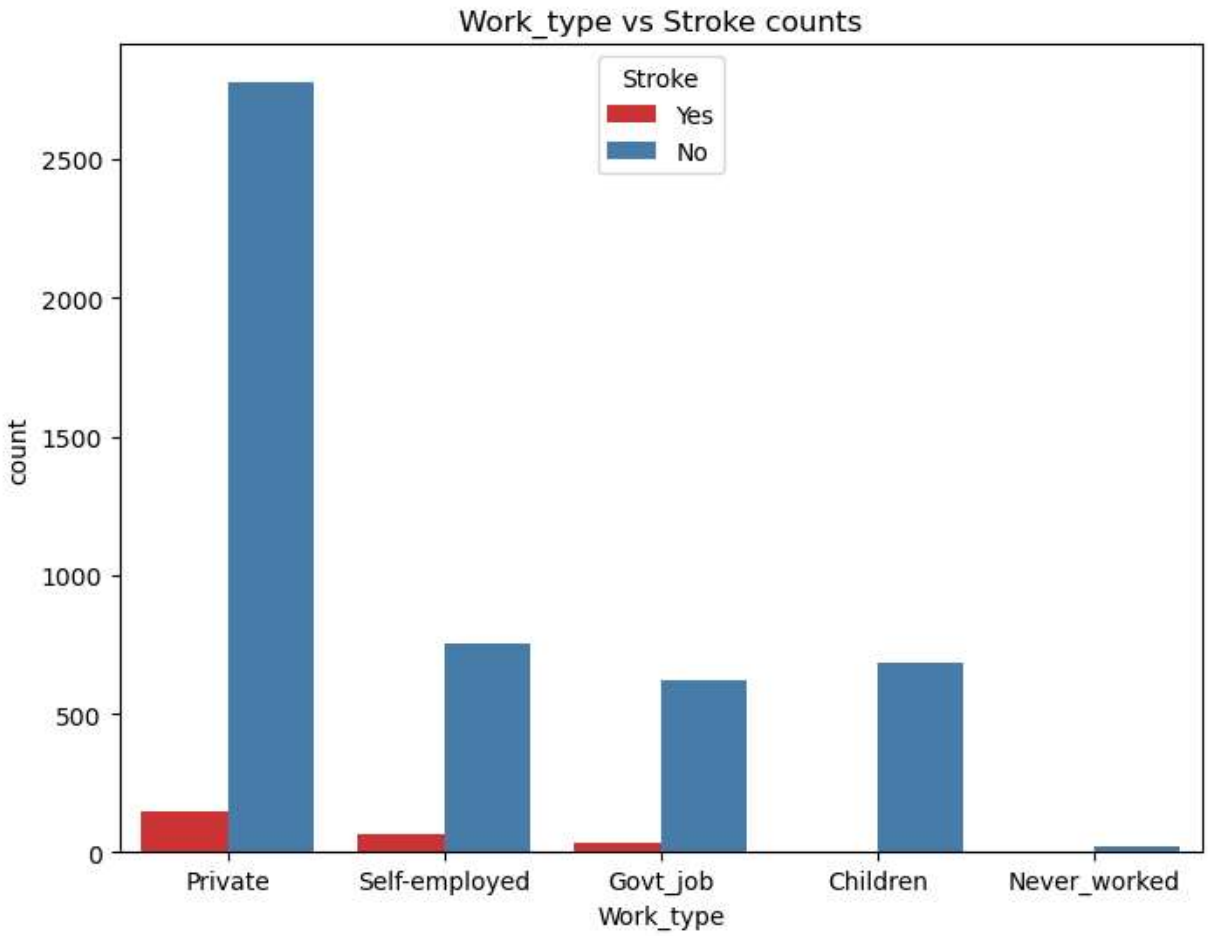
Out[17]:

Stroke	No	Yes
Ever_married		
No	0.983485	0.016515
Yes	0.934387	0.065613
All	0.951262	0.048738

- Observation:The married persons got the majority of stroke possibility.

Work_type

In [18]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Work_type'], hue = df['Stroke'])
ax.set_title('Work_type vs Stroke counts')
plt.show()
```



In [19]:
```
pd.crosstab(
    index = df['Work_type'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

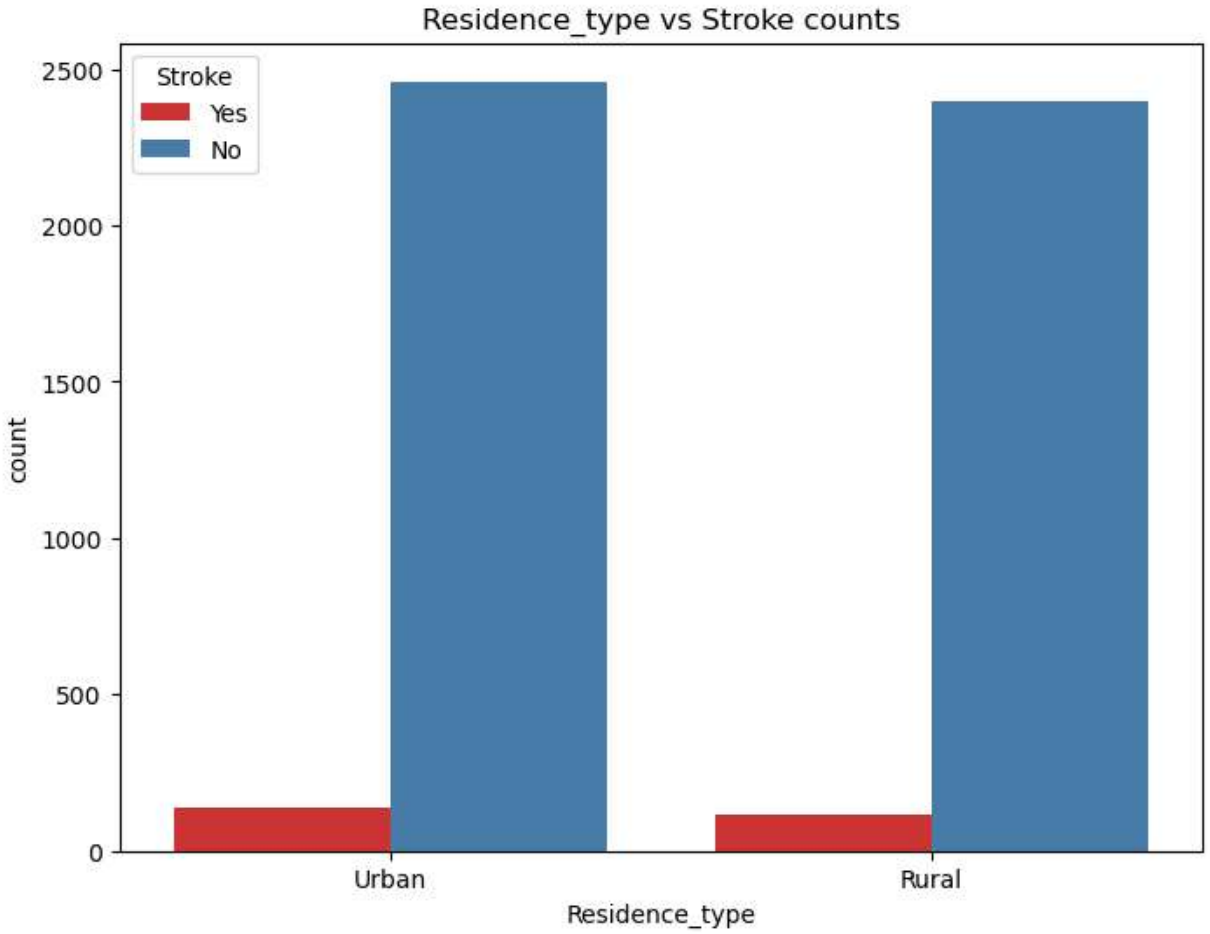
Out[19]:

Stroke	No	Yes
Work_type		
Children	0.997089	0.002911
Govt_job	0.949772	0.050228
Never_worked	1.000000	0.000000
Private	0.949042	0.050958
Self-employed	0.920635	0.079365
All	0.951262	0.048738

- Observation:Self-employed persons got the maximum number of stroke along with them private job persons are second highest affected persons.

Residence_type vs Stroke counts

In [20]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Residence_type'], hue = df['Stroke'])
ax.set_title('Residence_type vs Stroke counts')
plt.show()
```



In [21]:
```
pd.crosstab(
    index = df['Residence_type'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

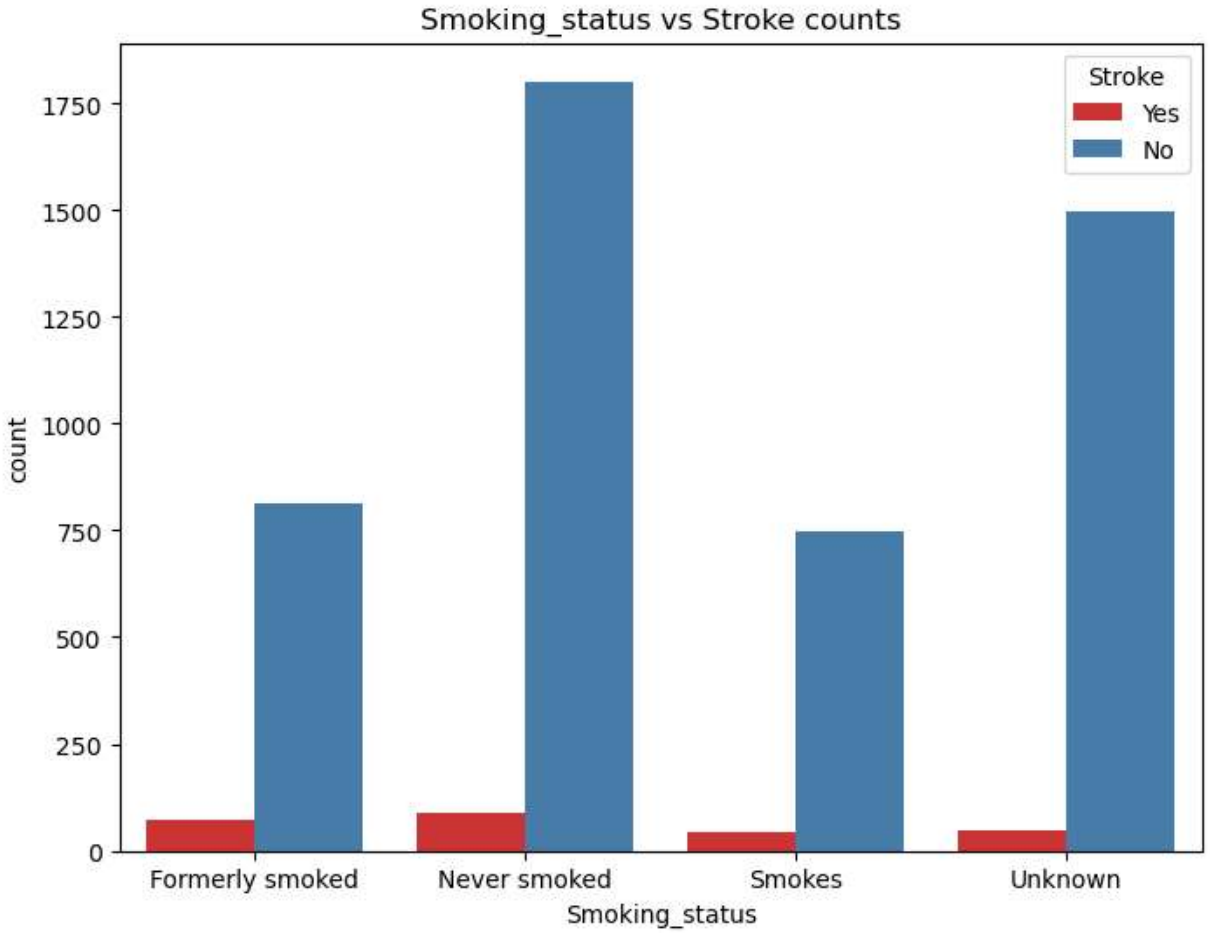
Out[21]:

Stroke	No	Yes
Residence_type		
Rural	0.954636	0.045364
Urban	0.947997	0.052003
All	0.951262	0.048738

- Observation:Urban people are mostly affected by brain stroke than the rural.

Smoking_status vs Stroke counts

In [22]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Smoking_status'], hue = df['Stroke'])
ax.set_title('Smoking_status vs Stroke counts')
plt.show()
```



In [23]:
```
pd.crosstab(
    index = df['Smoking_status'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

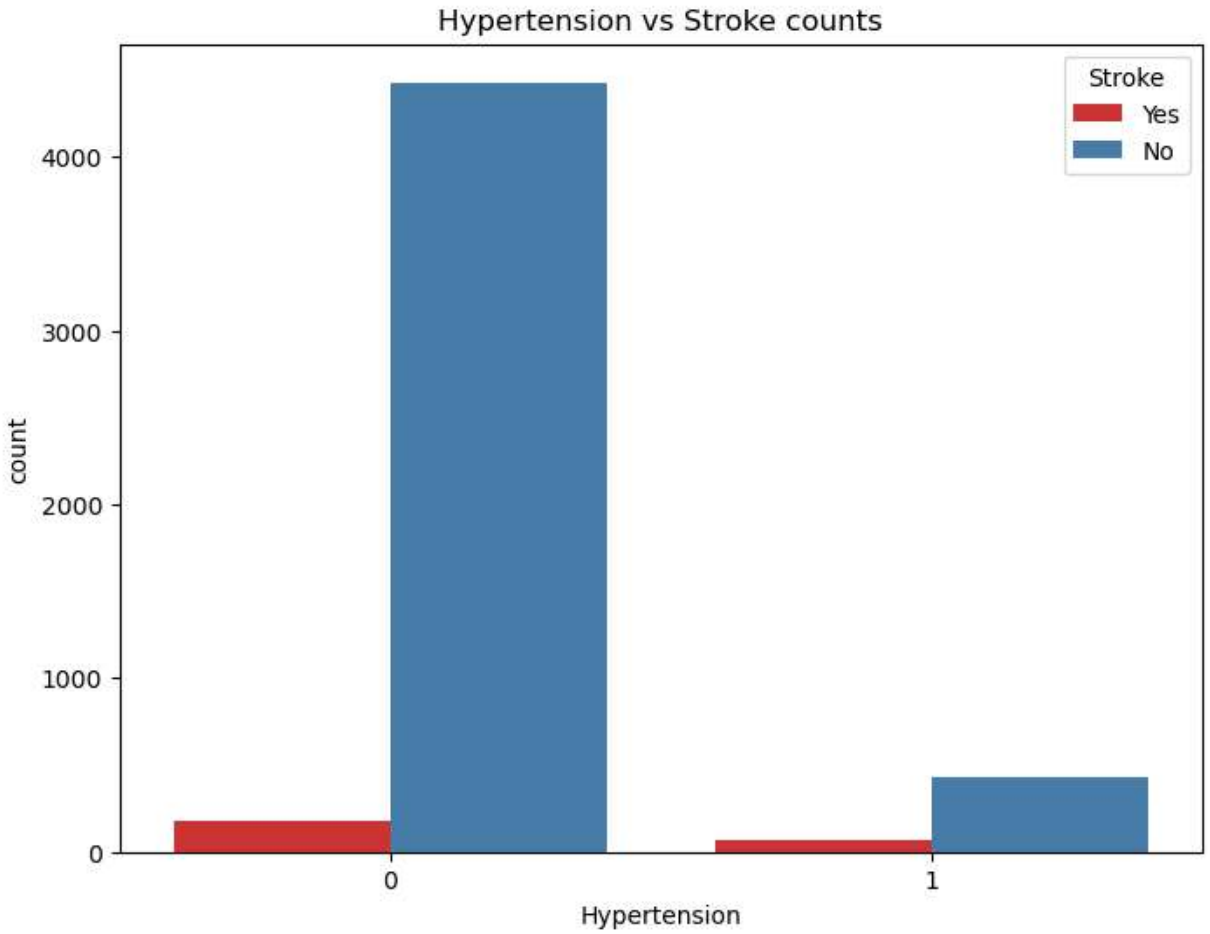
Out[23]:

Stroke	No	Yes
Smoking_status		
Formerly smoked	0.920814	0.079186
Never smoked	0.952431	0.047569
Smokes	0.946768	0.053232
Unknown	0.969560	0.030440
All	0.951262	0.048738

- Observation:Formerly smoked person are affected higher than the other category persons.

Hypertension vs Stroke counts

In [24]:
```
plt.figure(figsize = (8, 6))
sns.set_palette('Set1')
ax = sns.countplot(df['Hypertension'], hue = df['Stroke'])
ax.set_title('Hypertension vs Stroke counts')
plt.show()
```



In [25]:
```
pd.crosstab(
    index = df['Hypertension'],
    columns = df['Stroke'],
    margins = True,
    normalize = 'index',
)
```

Out[25]:

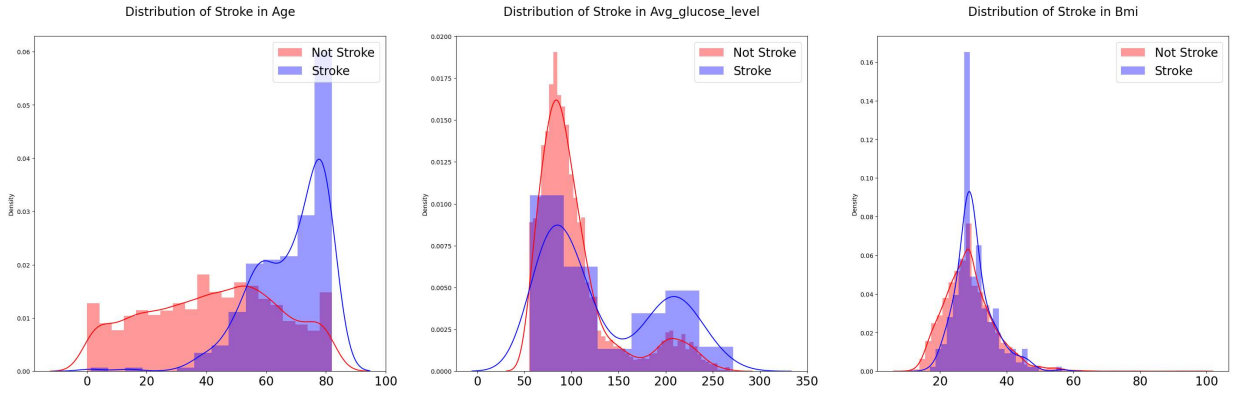
Stroke	No	Yes
Hypertension		
0	0.960312	0.039688
1	0.867470	0.132530
All	0.951262	0.048738

- Observation:The person having hypertension are getting more chance to get affected bt brain stroke.

Subplots

- The subplot() function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the first and second argument.
- The third argument represents the index of the current plot.
- The subplot is a side story that exists within the main plot.

In [26]:
```
cont_features = ['Age', 'Avg_glucose_level', 'Bmi']
stroke = df['Stroke'] == 'Yes'

fig, axs = plt.subplots(ncols = 3, nrows = 1, figsize = (20, 10))
plt.subplots_adjust(right = 1.5)

for i, feature in enumerate(cont_features):
    sns.distplot(df[~stroke][feature], label = 'Not Stroke', hist = True, color =
    sns.distplot(df[stroke][feature], label = 'Stroke', hist = True, color = 'blu

    axs[i].set_xlabel('')
    axs[i].tick_params(axis = 'x', labelsize = 20)

    axs[i].legend(loc = 'upper right', prop = {'size': 20})
    axs[i].set_title('Distribution of Stroke in {}'.format(feature), size = 20, y

plt.show()
```



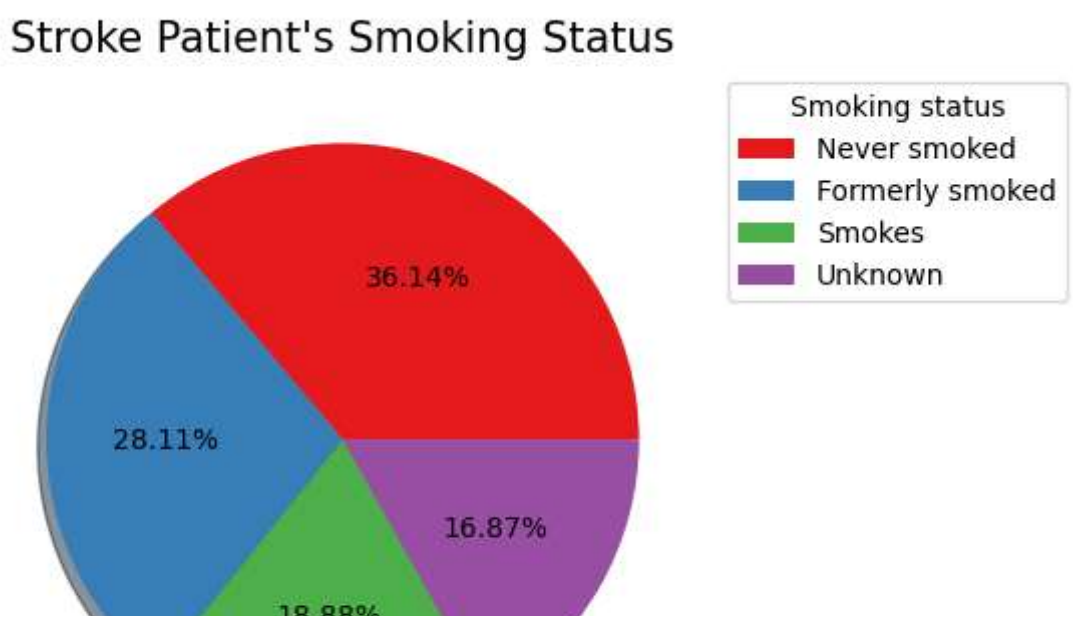
- Observation:

Pie chart

- Pie charts can be used to show percentages of a whole, and represents percentages at a set point in time.

- Graph can be created proportionally to the quantity it needs to represent.
- Displays multiple classes of data in one chart

In [27]:
```
labels = "Never smoked","Formerly smoked","Smokes","Unknown"
size = 0.5
wedges, texts, autotexts = plt.pie([df[df["Stroke"] == 'Yes']["Smoking_status"].v
                                    df[df["Stroke"] == 'Yes']["Smoking_status"].v
                                    df[df["Stroke"] == 'Yes']["Smoking_status"].v
                                    df[df["Stroke"] == 'Yes']["Smoking_status"].v

                                    autopct = "%.2f%%",
                                    shadow = True)

plt.legend(wedges, labels, title = "Smoking status", bbox_to_anchor=(1,1))
plt.title("\nStroke Patient's Smoking Status",fontsize = 15)
plt.rcParams["figure.figsize"] = (10, 5)
plt.show()
```

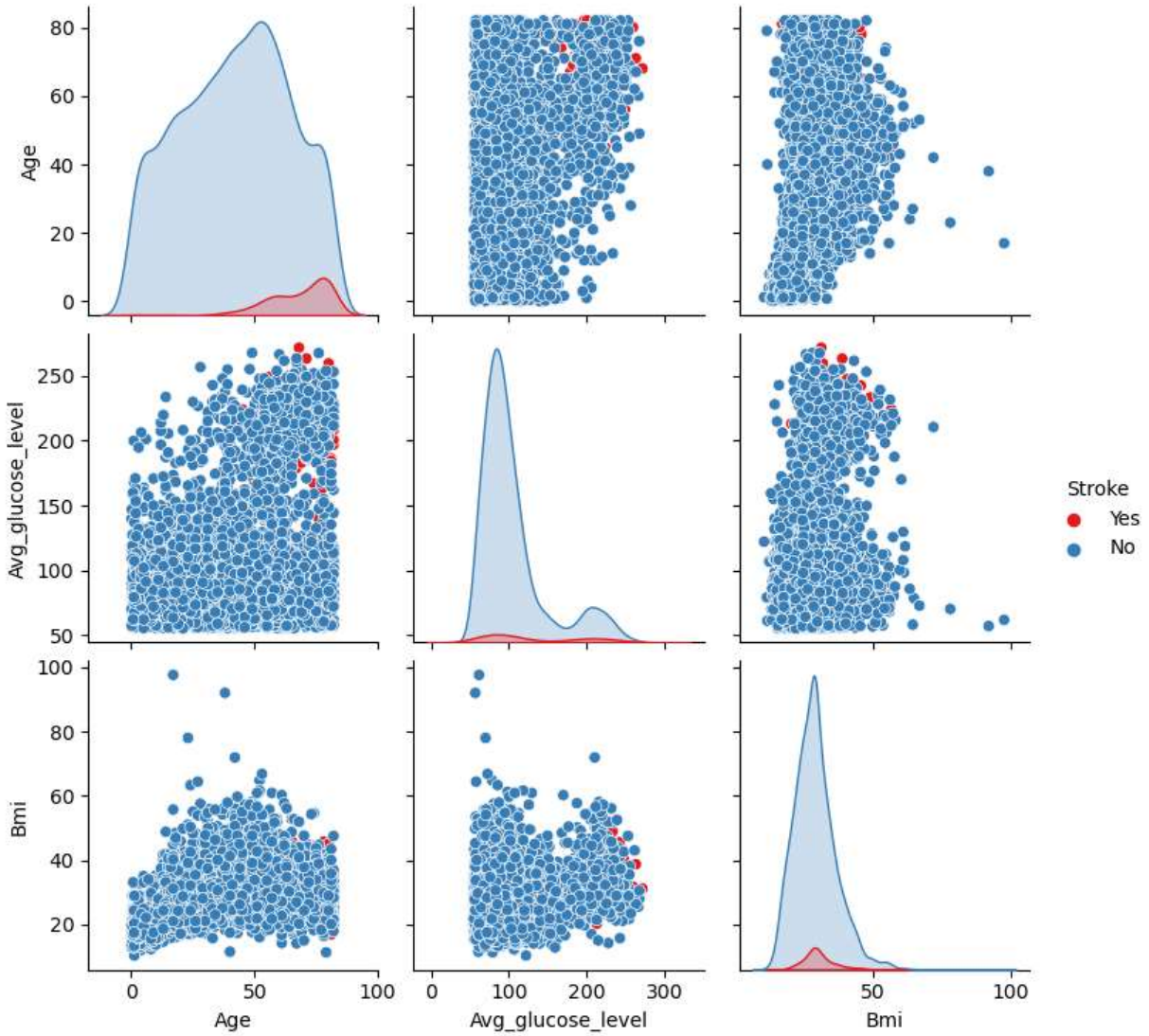


- Observation:Through the pie chart we can see the how many percent peoples are neversmoked,formerly smoked,smokes,unknown.

Pairplot

- Pairplots can be useful for exploring the relationships between variables in a dataset and identifying patterns or outliers.
- It is allows you to visualize the relationship between multiple variables in a dataset.

In [28]:
```
sns.set_palette('Set1')
plt = df[['Age','Avg_glucose_level','Bmi','Stroke']]
plt = sns.pairplot(plt, hue = 'Stroke')
```

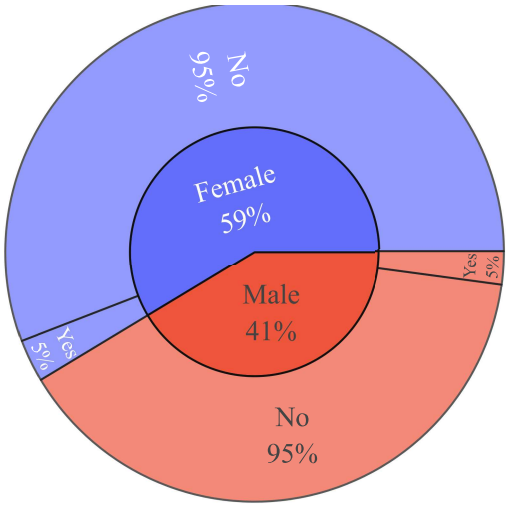


- Observation:By using pairplot we can clearly see the relationship between age,avg_glucose_level,bmi and the outliers.

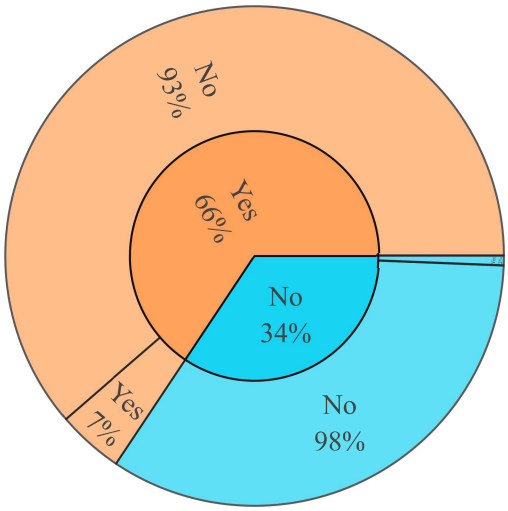
Sunburst

- The sunburst chart is ideal for displaying hierarchical data.
- The innermost circle represents the root of the hierarchy, and each level of the hierarchy is represented by a ring that surrounds the innermost circle.
- A sunburst chart, also known as a radial tree map or multi-level pie chart,

In [29]:
```
gender_stroke_df = df.groupby(['Gender', 'Stroke']).size().reset_index().rename(c

hypertension_stroke_df = df.groupby(['Hypertension', 'Stroke']).size().reset_inde

married_stroke_df = df.groupby(['Ever_married', 'Stroke']).size().reset_index().r

work_type_stroke_df = df.groupby(['Work_type', 'Stroke']).size().reset_index().re

residence_stroke_df = df.groupby(['Residence_type', 'Stroke']).size().reset_index

smoking_stroke_df = df.groupby(['Smoking_status', 'Stroke']).size().reset_index()

## Creating Sunburst Figures
sb1 = px.sunburst(gender_stroke_df, values = 'count', path = ['Gender', 'Stroke']
sb2 = px.sunburst(hypertension_stroke_df, values='count', path = ['Hypertension',

sb3 = px.sunburst(married_stroke_df, values = 'count', path = ['Ever_married', 'S
sb4 = px.sunburst(work_type_stroke_df, values = 'count', path = ['Work_type', 'St

sb5 = px.sunburst(residence_stroke_df, values = 'count', path = ['Residence_type'
sb6 = px.sunburst(smoking_stroke_df, values = 'count', path = ['Smoking_status',

## Subplots
fig = make_subplots(rows = 3, cols = 2, specs = [
    [{"type": "sunburst"}, {"type": "sunburst"}],
    [{"type": "sunburst"}, {"type": "sunburst"}],
    [{"type": "sunburst"}, {"type": "sunburst"}]
], subplot_titles = ("Gender and Stroke", "Hypertension and Stroke",
                     "Married and Stroke", "Work Type and Stroke",
                     "Residence and Stroke", "Smoking and Stroke"))

## Plotting Figures
fig.add_trace(sb1.data[0], row = 1, col = 1)
fig.add_trace(sb2.data[0], row = 1, col = 2)
fig.add_trace(sb3.data[0], row = 2, col = 1)
fig.add_trace(sb4.data[0], row = 2, col = 2)
fig.add_trace(sb5.data[0], row = 3, col = 1)
fig.add_trace(sb6.data[0], row = 3, col = 2)

fig.update_traces(textinfo="label+percent parent")

# Update title and height
fig.update_layout(title_text = "Stroke Sunbursts", title_x = 0.5, height = 1300,
        font = dict(
            family = "Rubik",
            size = 14)
)

fig.show()
```



Married and Stroke



- Observation:
- The chart shows the proportion of individuals who had a stroke and those who did not have a stroke within different subgroups of each categorical variable.
- The subgroups are represented by different levels of the hierarchy, with the outermost ring representing the categorical variable and each inner ring representing a subgroup of that variable.

Change in Values Type

In [30]:
```
df.Stroke.replace({'Yes':1,'No':0},inplace = True)
df.Gender.replace({'Male':0,'Female':1},inplace = True)
df.Ever_married.replace({'Yes':1,'No':0},inplace = True)
df.Work_type.replace({'Never_worked':0,'Private':1,'Self-employed':2,'Govt_job':
df.Residence_type.replace({'Urban':1,'Rural':2},inplace = True)
df.Smoking_status.replace({'Unknown':1,'Never smoked':2,'Smokes':3,'Formerly smok
```

In [31]:
```
df
```

Out[31]:

	Gender	Age	Hypertension	Heart_disease	Ever_married	Work_type	Residence_type	Avg_g
0	0	67.0	0	1	1	1	1	
1	1	61.0	0	0	1	2	2	
2	0	80.0	0	1	1	1	2	
3	1	49.0	0	0	1	1	1	
4	1	79.0	1	0	1	2	2	
...	...	...	...	...	...	...	...	
5104	1	80.0	1	0	1	1	1	
5105	1	81.0	0	0	1	2	1	
5106	1	35.0	0	0	1	2	2	
5107	0	51.0	0	0	1	1	2	
5108	1	44.0	0	0	1	3	1	

5109 rows × 11 columns

- Observation:Covert the string value to the integer value for classification progress.

Description of the dataset

In [32]:
```
df.describe()
```

Out[32]:

	Gender	Age	Hypertension	Heart_disease	Ever_married	Work_type	Residen
count	5109.000000	5109.000000	5109.000000	5109.000000	5109.000000	5109.000000	5109
mean	0.586025	43.229986	0.097475	0.054022	0.656293	1.816598	1
std	0.492592	22.613575	0.296633	0.226084	0.474991	1.105722	0
min	0.000000	0.080000	0.000000	0.000000	0.000000	0.000000	1
25%	0.000000	25.000000	0.000000	0.000000	0.000000	1.000000	1
50%	1.000000	45.000000	0.000000	0.000000	1.000000	1.000000	1
75%	1.000000	61.000000	0.000000	0.000000	1.000000	3.000000	2
max	1.000000	82.000000	1.000000	1.000000	1.000000	4.000000	2

- Observation:This method computes and displays summary statistics for a Python dataframe.

Correlation between the data

In [33]:
```
df.corr().style.background_gradient(cmap = 'GnBu')
```

Out[33]:

	Gender	Age	Hypertension	Heart_disease	Ever_married	Work_type	Re
Gender	1.000000	0.027752	-0.021223	-0.085685	0.030171	-0.063222	
Age	0.027752	1.000000	0.276367	0.263777	0.679084	-0.395002	
Hypertension	-0.021223	0.276367	1.000000	0.108292	0.164187	-0.069635	
Heart_disease	-0.085685	0.263777	0.108292	1.000000	0.114601	-0.054334	
Ever_married	0.030171	0.679084	0.164187	0.114601	1.000000	-0.353760	
Work_type	-0.063222	-0.395002	-0.069635	-0.054334	-0.353760	1.000000	
Residence_type	-0.006105	-0.014031	0.007980	-0.003045	-0.005988	-0.007830	
Avg_glucose_level	-0.054722	0.238323	0.174540	0.161907	0.155329	-0.064481	
Bmi	0.025605	0.325861	0.160151	0.038865	0.335564	-0.337697	
Smoking_status	-0.008920	0.360507	0.114815	0.092293	0.304471	-0.281901	
Stroke	-0.009081	0.245239	0.127891	0.134905	0.108299	-0.054539	

- Observation:corr() is used to find the pairwise correlation of all columns in the pandas dataframe in Python.

Separating the dataset

In [34]:
```
x = df.drop(['Stroke'],axis = 1).values
y = df['Stroke'].values
```

- Observation:Separating the dataset into x and y.

In [35]:
```
from sklearn.model_selection import train_test_split
```

In [141]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_stat
```

In [142]:
```
print(f'The aspect of x train dataset is',x_train.shape)
print(f'The aspect of x test dataset is',x_test.shape)
print(f'The aspect of y train dataset is',y_train.shape)
print(f'The aspect of y test dataset is',y_test.shape)
```

```
The aspect of x train dataset is (3831, 10)
The aspect of x test dataset is (1278, 10)
The aspect of y train dataset is (3831,)
The aspect of y test dataset is (1278,)
```

- Observation:
- Splitting the data for train and test.
- where 75% of data for train and 25% of data for test.

Feature scaling the values

- Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.

In [143]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
x_train
x_test
```

Out[143]:
```
array([[-1.18210478, -1.57009884, -0.32891684, ..., -0.97974991,
        -1.28148706, -1.1275382 ],
       [-1.18210478, -1.83675982, -0.32891684, ..., -0.2606707 ,
        -0.72922573, -1.1275382 ],
       [ 0.8459487 , -1.79231632, -0.32891684, ..., -0.2812601 ,
        -1.76800301, -1.1275382 ],
       ...,
       [-1.18210478, -1.79231632, -0.32891684, ..., -0.51903444,
        -1.42612694, -1.1275382 ],
       [-1.18210478,  0.47430202, -0.32891684, ...,  0.20557955,
         1.47981963,  1.71679817],
       [ 0.8459487 ,  0.29652803, -0.32891684, ..., -0.09728386,
         0.50678775, -0.17942607]])
```

- Observation:
- The features of the data is transforming to the mean of each feature is 0 and the variance is 1.
- By transforming the data to have zero mean and unit variance, the algorithm is more likely to treat all features equally and converge faster to the optimal solution.

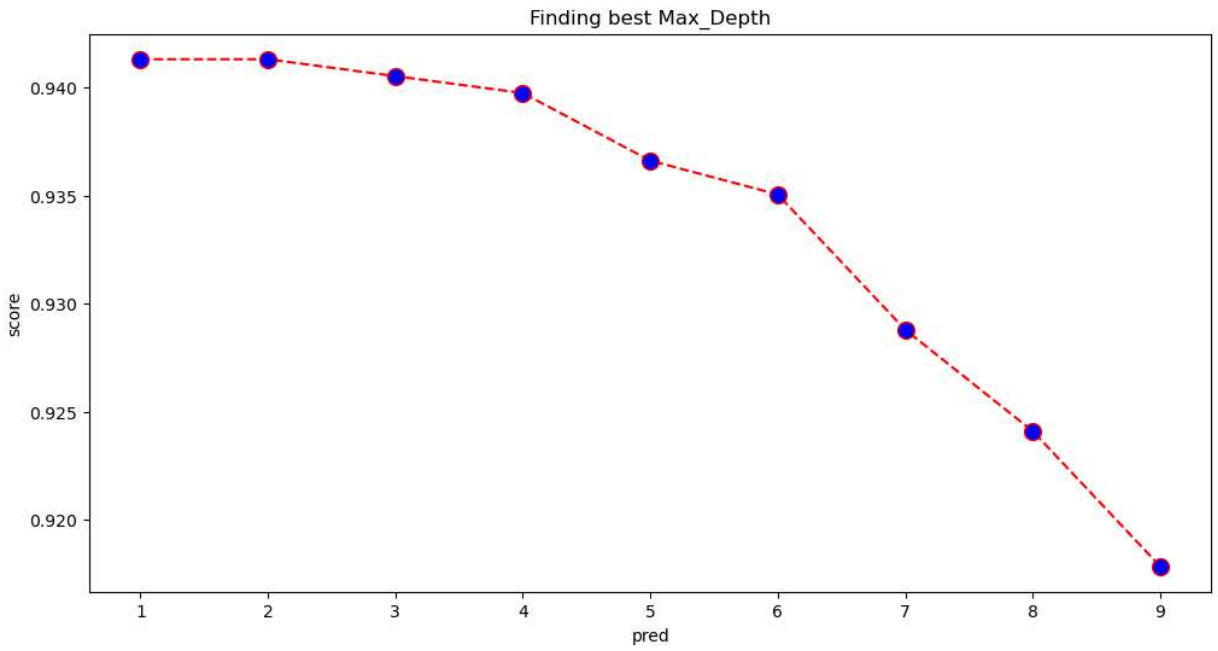
Decision Tree

- The tree is built from a set of rules that split the input space into regions, with each region corresponding to a different class label.
- The prediction for a new instance is made by traversing the tree from the root to a leaf node, and taking the class label associated with that node as the final prediction.

In [144]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [145]:
```
accuracy = []
for i in range(1, 10):
    model_DC = DecisionTreeClassifier(max_depth = i, random_state = 0)
    model_DC.fit(x_train,y_train)
    pred = model_DC.predict(x_test)
    score = accuracy_score(y_test, pred)
    accuracy.append(score)

plt.figure(figsize = (12, 6))
plt.plot(range(1, 10), accuracy, color = 'red', linestyle = 'dashed', marker = 'o
         markerfacecolor='blue', markersize = 10)
plt.title('Finding best Max_Depth')
plt.xlabel('pred')
plt.ylabel('score')
```

Out[145]:
```
Text(0, 0.5, 'score')
```



- Finding the max_depth value.

In [146]:
```
from sklearn.tree import DecisionTreeClassifier
model_DC = DecisionTreeClassifier(max_depth = 3)
model_DC.fit(x_train,y_train)
```

Out[146]:
```
DecisionTreeClassifier(max_depth=3)
```

In [147]:
```
DC = model_DC.score(x_test,y_test)*100
DC
```

Out[147]:
```
94.05320813771519
```

- The accuracy of the model by applying DecisionTree classifeier is 94.05.

In [148]:
```
y_predict = model_DC.predict(x_test)
y_predict
```

Out[148]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [149]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[149]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

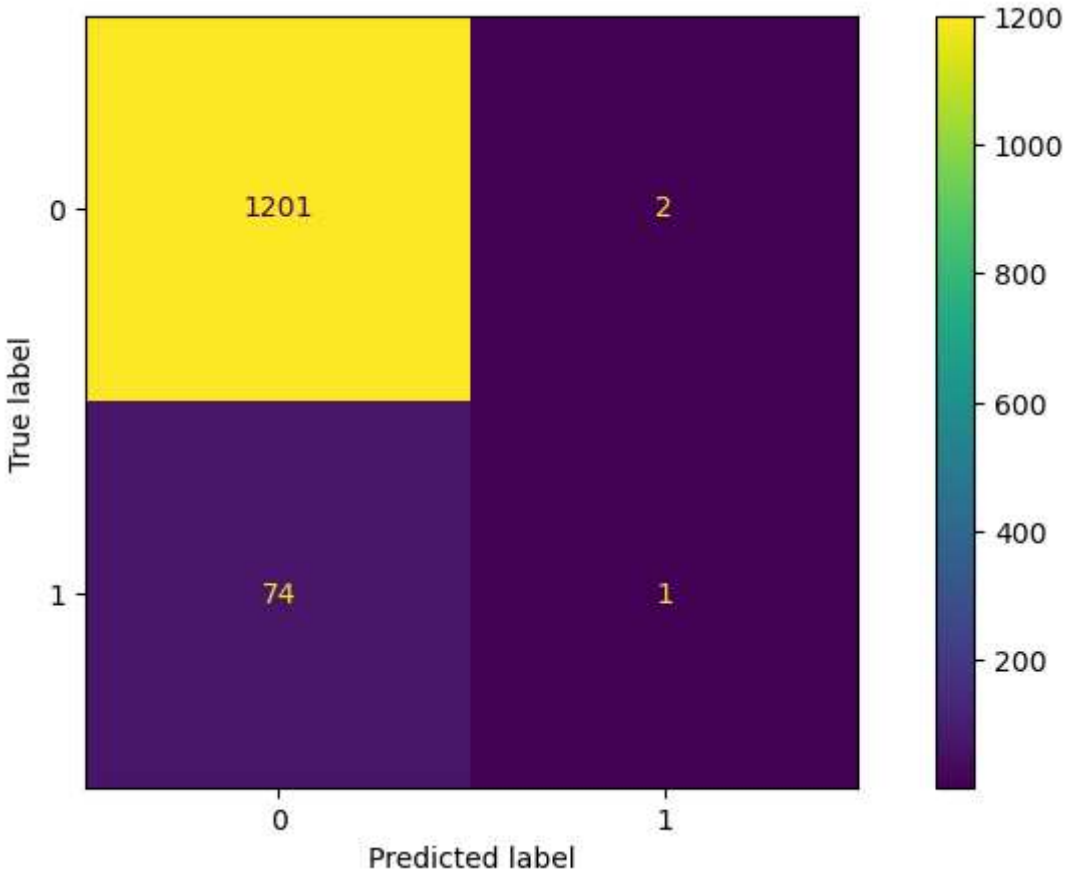
1278 rows × 2 columns

- Comparing the Actual value and the Predicted value.

In [150]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[150]:
```
array([[1201,    2],
       [  74,    1]], dtype=int64)
```

In [151]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_DC,x_test,y_test)
plt.show()
```



In [152]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1203
           1       0.33      0.01      0.03        75

    accuracy                           0.94      1278
   macro avg       0.64      0.51      0.50      1278
weighted avg       0.91      0.94      0.91      1278
```

Random Forest

- It creates multiple trees and combines their outputs to make a final prediction.
- Each tree is built from a random subset of the training data, and a random subset of the input features.
- The final prediction is made by aggregating the predictions from all trees the majority vote.

In [153]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [154]:
```
model_RF = RandomForestClassifier()
model_RF.fit(x_train,y_train)
```

Out[154]:
```
RandomForestClassifier()
```

In [155]:
```
RF = model_RF.score(x_test,y_test)*100
RF
```

Out[155]:
```
93.97496087636932
```

- The accuracy of the model by applying RandomForest classifeier is 93.97.

In [156]:
```
y_predict = model_RF.predict(x_test)
y_predict
```

Out[156]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [157]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[157]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

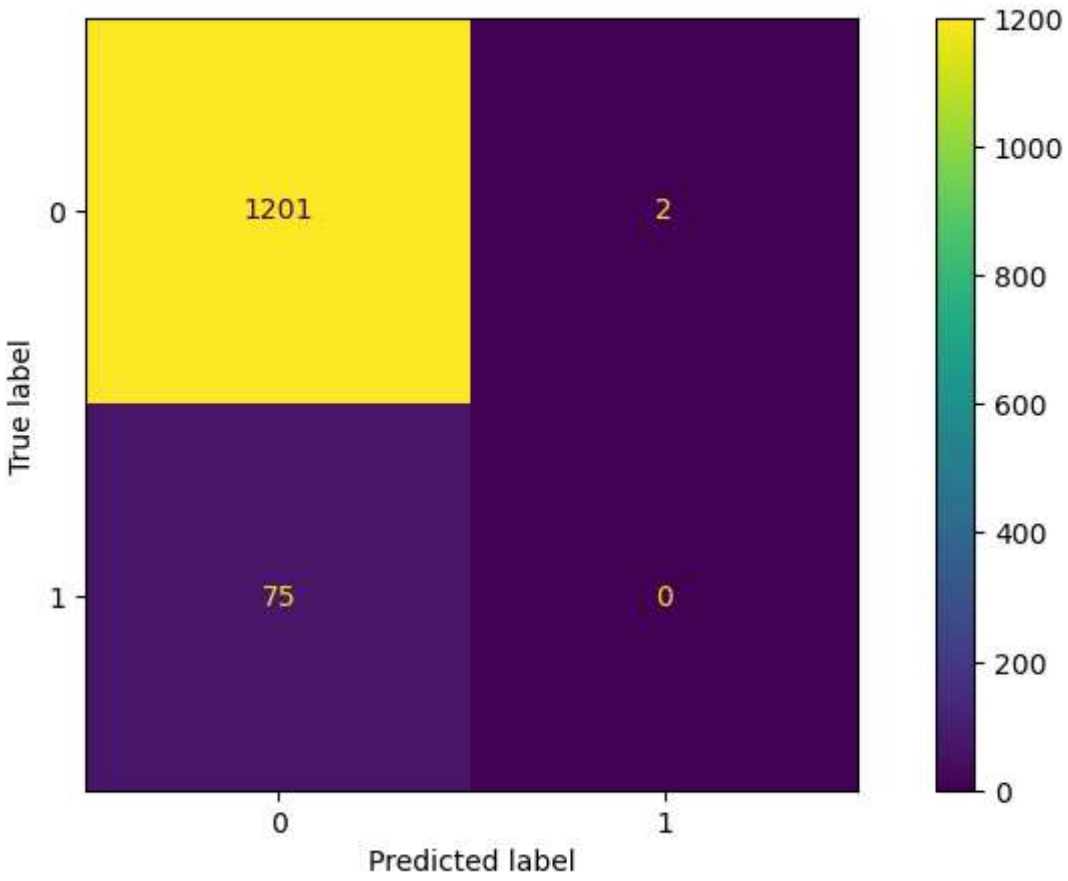
1278 rows × 2 columns

- Comparing the Actual value and the Predicted value.

In [158]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[158]:
```
array([[1201,    2],
       [  75,    0]], dtype=int64)
```

In [159]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_RF,x_test,y_test)
plt.show()
```



In [160]:
```
from sklearn.metrics import accuracy_score
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1203
           1       0.00      0.00      0.00        75

    accuracy                           0.94      1278
   macro avg       0.47      0.50      0.48      1278
weighted avg       0.89      0.94      0.91      1278
```

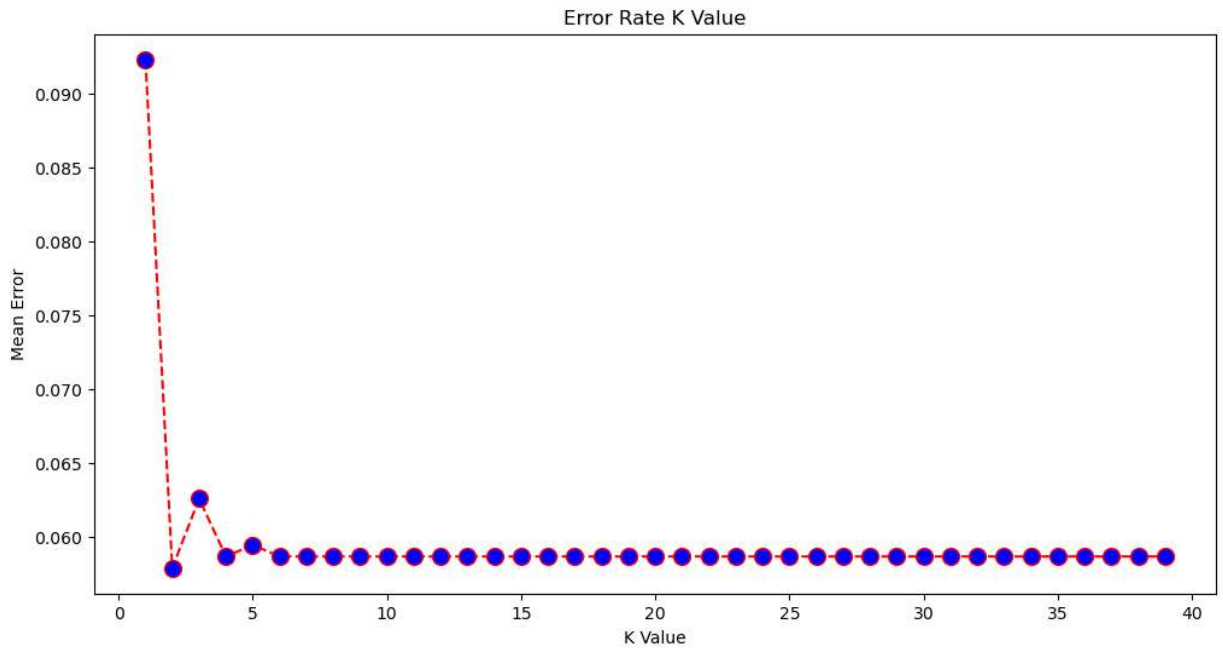
KNN

- The algorithm works by storing the training instances and their corresponding class labels.
- To make a prediction for a new instance, the KNN algorithm finds the K nearest training instances to the new instance and outputs the class label that is most frequent among those K nearest neighbors.

In [161]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [162]:
```
error = []
for i in range(1, 40):
    model_KNN = KNeighborsClassifier(n_neighbors=i)
    model_KNN.fit(x_train, y_train)
    pred_i = model_KNN.predict(x_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize = (12, 6))
plt.plot(range(1, 40),error,color = 'red', linestyle = 'dashed', marker = 'o',
         markerfacecolor = 'blue', markersize = 10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Out[162]:
```
Text(0, 0.5, 'Mean Error')
```



- Finding the neighbour value.

In [163]:
```
from sklearn.neighbors import KNeighborsClassifier
model_KNN = KNeighborsClassifier(n_neighbors=2)
model_KNN.fit(x_train,y_train)
```

Out[163]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [164]:
```
KNN=model_KNN.score(x_test,y_test)*100
KNN
```

Out[164]:
```
94.20970266040689
```

- The accuracy of the model by applying KNN classifeier is 94.20.

In [165]:
```
y_predict = model_KNN.predict(x_test)
y_predict
```

Out[165]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [166]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[166]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

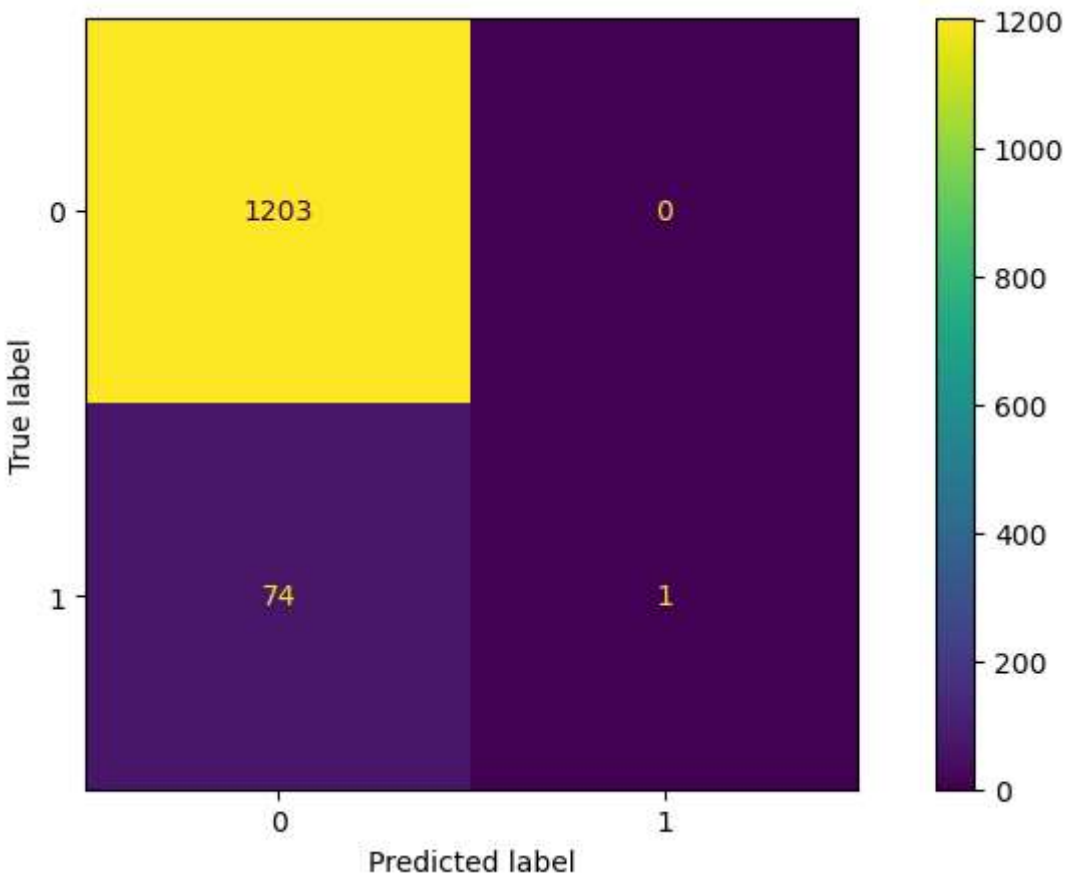
1278 rows × 2 columns

- Comparing the Actual value and the Predicted value.

In [167]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[167]:
```
array([[1203,    0],
       [  74,    1]], dtype=int64)
```

In [168]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_KNN,x_test,y_test)
plt.show()
```



In [169]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1203
           1       1.00      0.01      0.03        75

    accuracy                           0.94      1278
   macro avg       0.97      0.51      0.50      1278
weighted avg       0.95      0.94      0.91      1278
```

Naive Bayes

- The algorithm requires computing the prior probabilities of the class labels, which can be

estimated from the training data, and the likelihoods of the features given each class, which can be modeled using probability distributions.

- It is based on Bayes' theorem, which states that the probability of a class label given an input feature vector can be computed from the prior probability of the class label and the likelihood of the features given the class label.

In [170]:
```
from sklearn.naive_bayes import GaussianNB
model_GB = GaussianNB()
model_GB.fit(x_train,y_train)
```

Out[170]:
```
GaussianNB()
```

In [171]:
```
NB = model_GB.score(x_test,y_test)*100
NB
```

Out[171]:
```
86.46322378716745
```

- The accuracy of the model by applying Naive Bayes classifeier is 86.46.

In [172]:
```
y_predict = model_GB.predict(x_test)
y_predict
```

Out[172]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [173]:
```
y_predict = model_GB.predict(x_test)
y_predict
```

Out[173]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [174]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[174]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	1
4	0	1
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

1278 rows × 2 columns

- Comparing the Actual value and the Predicted value.

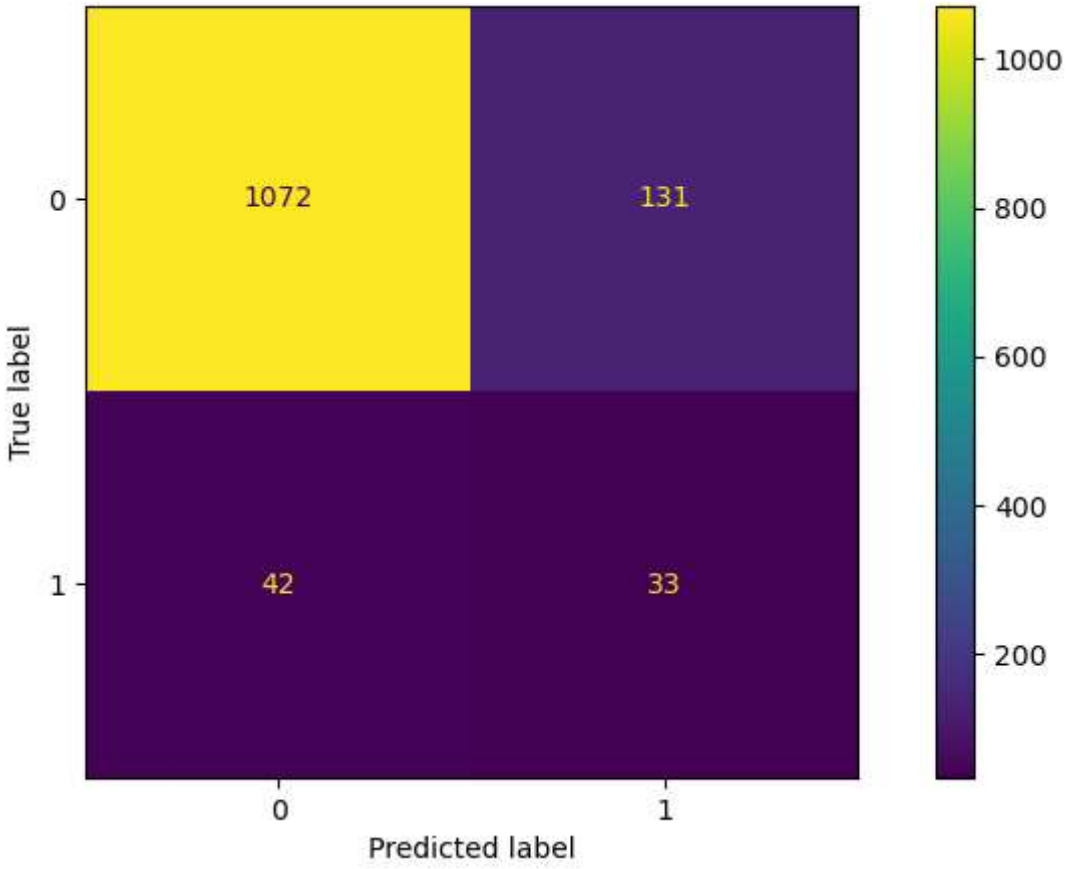
In [175]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[175]:
```
array([[1072,  131],
       [  42,   33]], dtype=int64)
```

In [176]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_predict)*100
```

Out[176]:
```
86.46322378716745
```

In [177]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_GB,x_test,y_test)
plt.show()
```



In [178]:
```
from sklearn.metrics import accuracy_score
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.89      0.93      1203
           1       0.20      0.44      0.28        75

    accuracy                           0.86      1278
   macro avg       0.58      0.67      0.60      1278
weighted avg       0.92      0.86      0.89      1278
```

SVM

- SVC works by finding a hyperplane that separates the classes in a high dimensional space, the hyperplane is chosen so that it has the maximum margin from the closest data points from each class.

In [179]:
```
from sklearn.svm import SVC
model_SVC = SVC()
model_SVC.fit(x_train,y_train)
```

Out[179]:
```
SVC()
```

In [180]:
```
SVC = model_SVC.score(x_test,y_test)*100
SVC
```

Out[180]:
```
94.13145539906104
```

- The accuracy of the model by applying SVC Classifier is 94.13.

In [181]:
```
y_predict = model_SVC.predict(x_test)
y_predict
```

Out[181]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [182]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[182]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

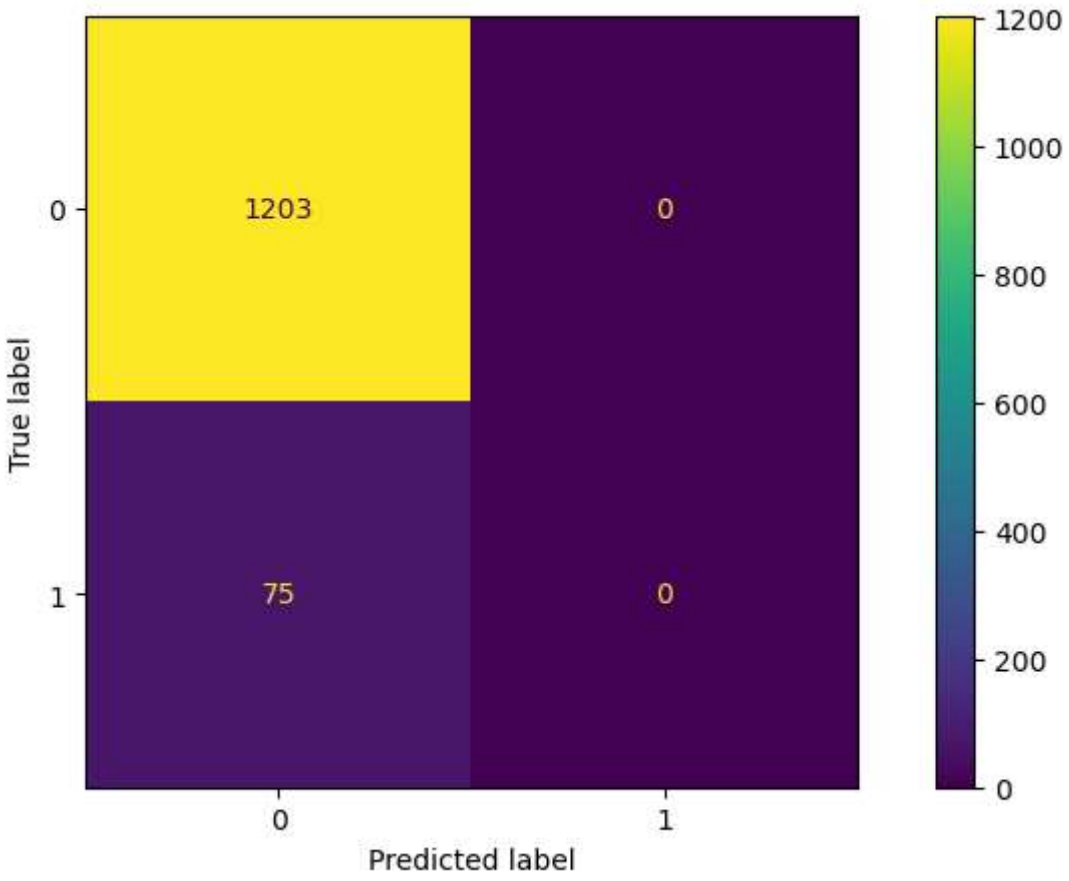
1278 rows × 2 columns

- Comparing the Actual value and the Predicted value.

In [183]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[183]:
```
array([[1203,    0],
       [  75,    0]], dtype=int64)
```

In [184]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_SVC,x_test,y_test)
plt.show()
```



In [185]:
```
from sklearn.metrics import accuracy_score
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1203
           1       0.00      0.00      0.00        75

    accuracy                           0.94      1278
   macro avg       0.47      0.50      0.48      1278
weighted avg       0.89      0.94      0.91      1278
```

AdaBoost

In [186]:
```
from sklearn.ensemble import AdaBoostClassifier
model_AB = AdaBoostClassifier(n_estimators = 50, learning_rate = 1)
model_AB.fit(x_train,y_train)
```

Out[186]:
```
AdaBoostClassifier(learning_rate=1)
```

In [187]:
```
AdaBoost=model_AB.score(x_test,y_test)*100
AdaBoost
```

Out[187]:
```
93.97496087636932
```

- The accuracy of the GaussianNB model after boosting is 93.97.

In [188]:
```
y_predict = model_AB.predict(x_test)
y_predict
```

Out[188]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [189]:
```
df = pd.DataFrame({'actual':y_test,'predicted':y_predict})
df
```

Out[189]:

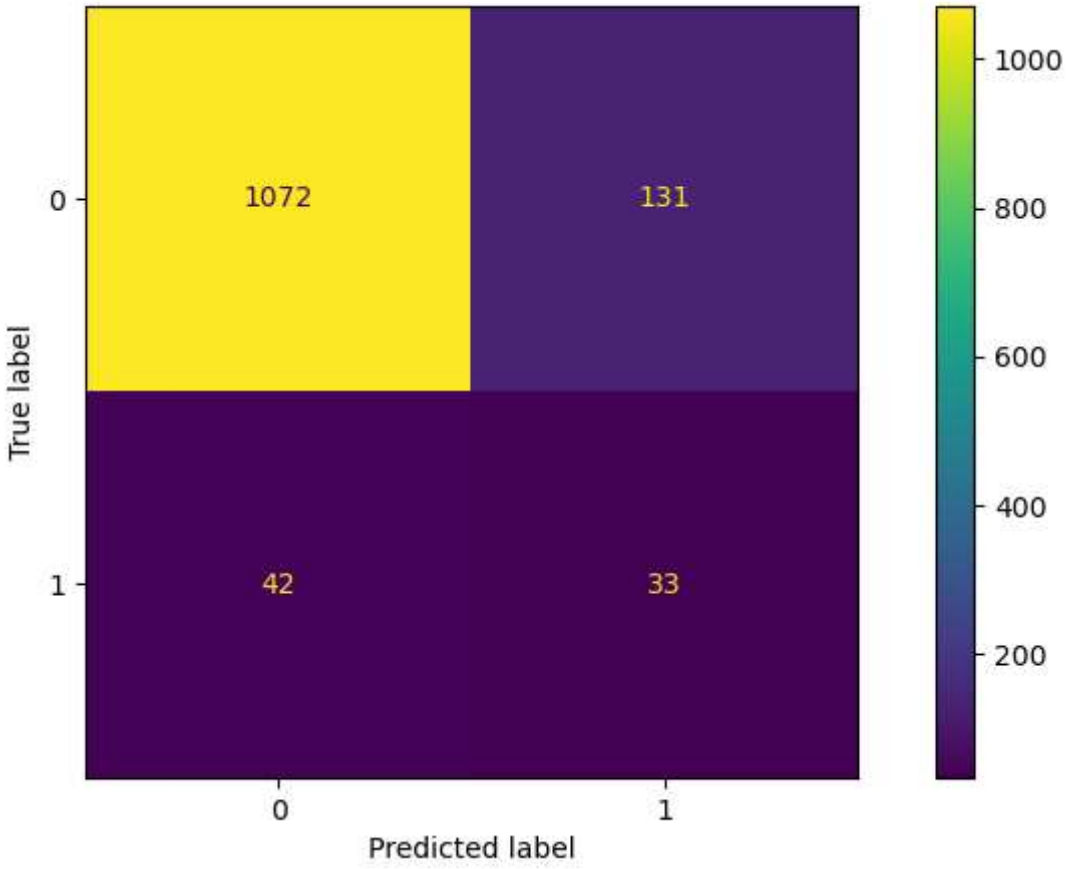
	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1
...	...	...
1273	0	0
1274	0	0
1275	0	0
1276	0	0
1277	0	0

1278 rows × 2 columns

In [190]:
```
from sklearn.metrics import confusion_matrix
performance = confusion_matrix(y_test,y_predict)
performance
```

Out[190]:
```
array([[1200,    3],
       [  74,    1]], dtype=int64)
```

In [191]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model_GB,x_test,y_test)
plt.show()
```



In [192]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))
```

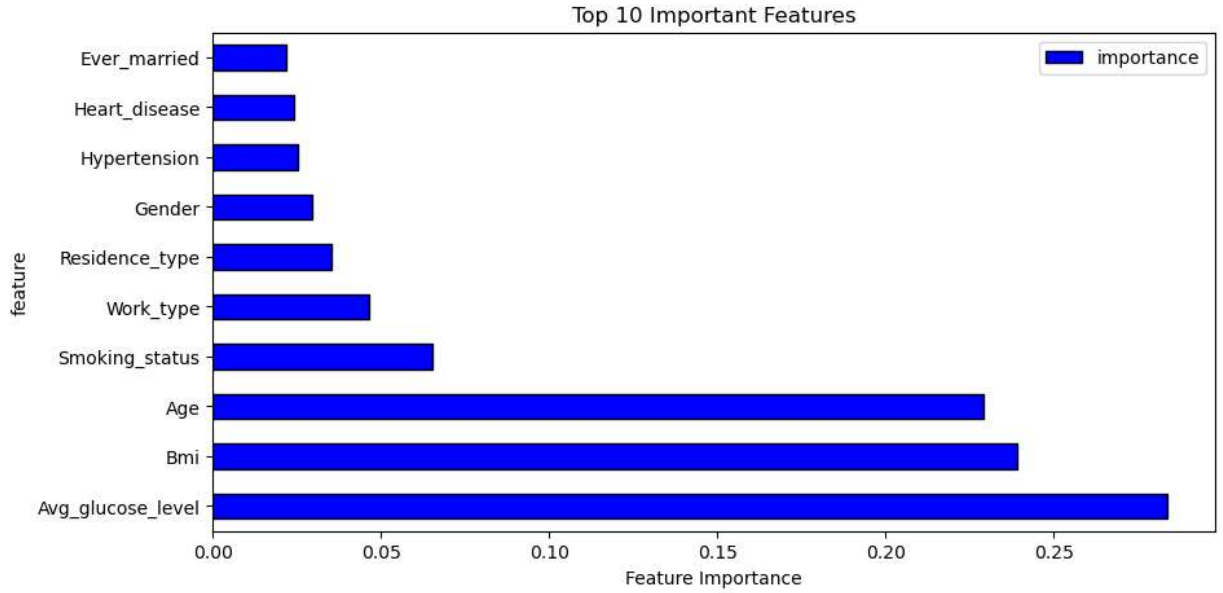
```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1203
           1       0.25      0.01      0.03        75

    accuracy                           0.94      1278
   macro avg       0.60      0.51      0.50      1278
weighted avg       0.90      0.94      0.91      1278
```

Feature Importance

In [193]:
```
from sklearn.ensemble import AdaBoostClassifier
importance=model_RF.feature_importances_
for i,v in enumerate(importance):
    print('Feature: %d, score: %.3f' % (i,v))
```

```
Feature: 0, score: 0.030
Feature: 1, score: 0.229
Feature: 2, score: 0.025
Feature: 3, score: 0.024
Feature: 4, score: 0.022
Feature: 5, score: 0.047
Feature: 6, score: 0.035
Feature: 7, score: 0.284
Feature: 8, score: 0.239
Feature: 9, score: 0.065
```

In [194]:
```
x = pd.DataFrame(x, columns=['Gender', 'Age', 'Hypertension', 'Heart_disease', 'E
importance = model_RF.feature_importances_

# store feature importances in a dataframe
feat_importances = pd.DataFrame({'feature': x.columns, 'importance': importance})

# plot the 10 largest feature importances
feat_importances.nlargest(10, 'importance').plot(kind='barh', x='feature', y='imp
plt.xlabel('Feature Importance')
plt.title('Top 10 Important Features')
plt.show()
```



- Observation:
- we can conclude that the top 3 significant features were maximum stroke achieved by Avg_glucose_level, Age and Bmi.

Comparision of the models

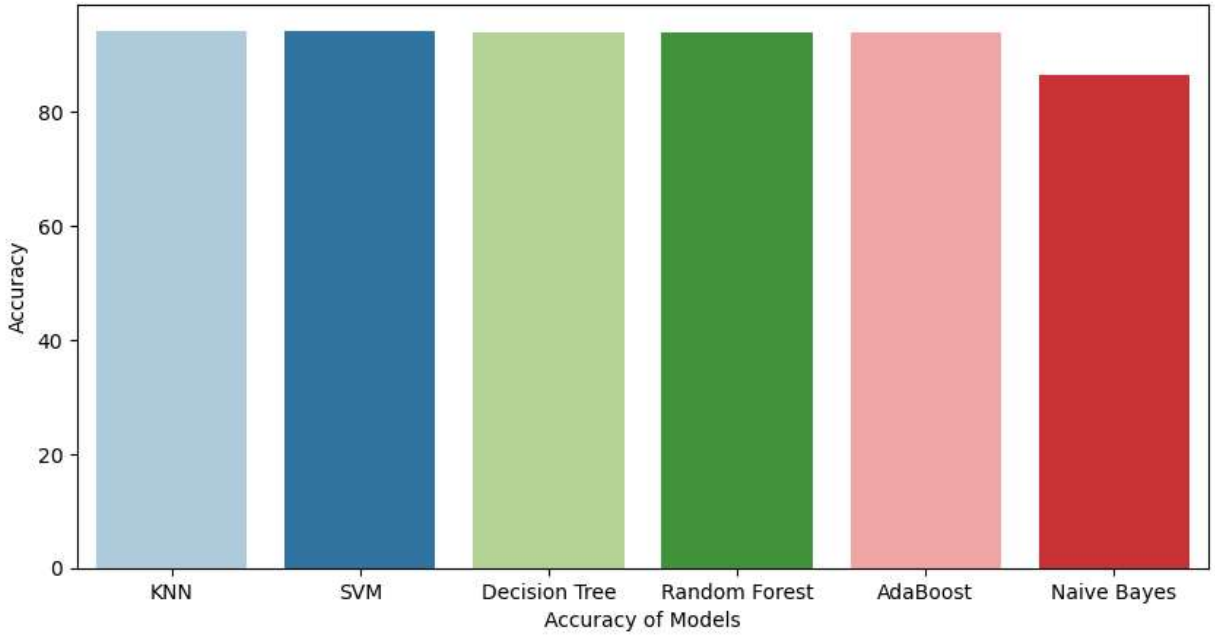
In [199]:
```
results=pd.DataFrame({
    'Models':['Decision Tree',
              'Random Forest',
              'KNN',
              'Naive Bayes',
              'SVM',
              'AdaBoost'],
    'Accuracy':[
                DC,
                RF,
                KNN,
                NB,
                SVC,
                AdaBoost]
})
result_df=results.sort_values(by='Accuracy',ascending=False)
result_df=result_df.reset_index(drop=True)
result_df
```

Out[199]:

	Models	Accuracy
0	KNN	94.209703
1	SVM	94.131455
2	Decision Tree	94.053208
3	Random Forest	93.974961
4	AdaBoost	93.974961
5	Naive Bayes	86.463224

- Observation:
- KNN algorithm yields the highest accuracy, 94.2%.
- Naive Bayes yeilds the lowest accuracy, 86.46%.

In [198]:
```
models = {'Model': ['Decision Tree',
                    'Random Forest',
                    'KNN',
                    'Naive Bayes',
                    'SVM',
                    'AdaBoost'],
          'Accuracy': [DC,
                       RF,
                       KNN,
                       NB,
                       SVC,
                       AdaBoost]}
models = pd.DataFrame(models)
models = models.sort_values(by='Accuracy', ascending=False)
plt.figure(figsize=(10,5))
sns.barplot(x='Model', y='Accuracy', data=models, palette="Paired")
plt.xlabel('Accuracy of Models', fontsize=10)
plt.show()
```



A Predictive System

In [197]:
```
input_data=(1,58,0,1,1,1,2,144.16,26,3)
input_data_as_numpy_array=np.asarray(input_data)
input_data_reshaped=input_data_as_numpy_array.reshape(1,-1)
prediction=model_AB.predict(input_data_reshaped)
print(prediction)

if prediction[0]==0:
    print('The person is not affected by Brain Stroke')
else:
    print('The person is affected by Brain Stroke')
```

```
[0]
The person is not affected by Brain Stroke
```

- Observation:
- By applying the model in real life scenario we got the result, that the person is not affected by Brain Stroke.

Conclusion

- KNN algorithm yields the highest accuracy, 94.2%. Any accuracy above 80% is considered good, but be careful because if your accuracy is extremely high, it may be too good to be true (an example of Overfitting). Thus, 94.2% is the ideal accuracy!
- Out of the 10 features we examined, the top 3 significant features that helped us classify between a positive & negative maximum stroke achieved by Avg_glucose _level, Age and Bmi .
- Our machine learning algorithm can now classify patients with Brain Stroke. Now we can properly thrombectomy, & get them the help they needs to recover. By detecting these features early, we may prevent worse symptoms from arising later.

In [ ]:
```
```